PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30 — A1

(11) International Publication Number: **WO 99/38095**

(43) International Publication Date: 29 July 1999 (29.07.99)

(21) International Application Number: PCT/NO99/00018

(22) International Filing Date: 25 January 1999 (25.01.99)

(30) Priority Data:
09/013,678 26 January 1998 (26.01.98) US
09/013,808 26 January 1998 (26.01.98) US

(71) Applicant: TELENOR AS [NO/NO]; Universitetsgaten 2, N–0164 Oslo (NO).

(72) Inventor: ANFINDSEN, Ole, J.; Lotterudveien 7, N–1912 Enebakk (NO).

(74) Agent: LANGFELDT, Jens, F., C.; Bryns Patentkontor a/s, P.O. Box 765, Sentrum, N–0106 Oslo (NO).

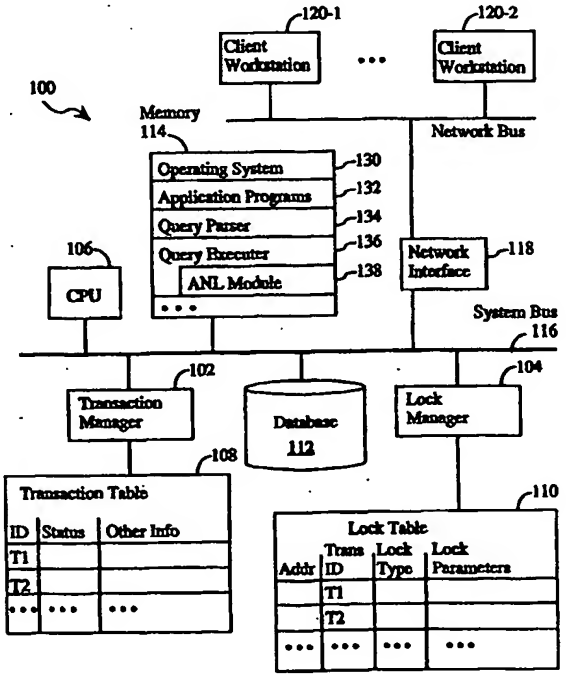
(81) Designated States: AU, CA, JP, NO, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: DATABASE MANAGEMENT SYSTEM AND METHOD FOR CONDITIONAL CONFLICT SERIALIZABILITY OF TRANSACTIONS AND FOR COMBINING META–DATA OF VARYING DEGREES OF RELIABILITY

(57) Abstract

A database management system (DBMS) is modified to provide improved concurrent usage of database objects, particularly when the system is executing long lived transactions. A subset of the transactions access database objects using parameterized read and parameterized write access modes. Each transaction using a parameterized write mode of access for a database objet specifies a write access mode, and a write access mode parameter, where the parameter indicates a data reliability classification. Each transaction using a parameterized read mode of access for a database object specifies a read access mode, and a read access mode parameter, where the parameter indicates one or more reliability classifications that are acceptable to the transaction. Whenever a transaction requests access to a specified database object, the DBMS generates a corresponding lock request for the object. If the lock request is parameterized lock request, a corresponding parameterized lock request is generated. A lock manager processes each lock request by checking to see if any outstanding, previously granted lock is a unconditionally conflicting or conditionally conflicting wit the requested lock. A query executer queries requesting access to information stored in the database (or other data processing tasks). The query executer includes annotated nullable logic for evaluating expressions containing at least one annotated value as an operand. The annotated nullable logic includes logic for combining annotated values, for comparing annotated values so as to generate annotated truth values, and for combining annotated truth values in accordance with a predefined set of rules.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

DATABASE MANAGEMENT SYSTEM AND METHOD FOR CONDITIONAL CONFLICT SERIALIZABILITY OF TRANSACTIONS AND FOR COMBINING META-DATA OF VARYING DEGREES OF RELIABILITY

The present invention relates generally to database management systems and transaction processing systems that utilize a lock manager for protecting database resources from simultaneous incompatible uses, and more particularly to a lock manager that allows greater concurrent use of resources than the lock managers in traditional transaction processing systems while diminishing the "ACID" properties of transactions only with respect to isolation between concurrent transactions. The present invention also relates to a database management system and method for utilizing and combining data, some of which is entirely reliable and some of which is less reliable, unknown or missing, using a methodology herein called annotated nullable logic for comparing and combining annotated data and truth values in a manner that preserves reliability category information associated with the data and truth values being compared and/or combined.

BACKGROUND OF THE INVENTION

The present invention is directed at the management of transactions in database management systems so as to enable greater concurrency, and therefore more efficient transaction execution, than is allowed by DBMS's requiring strict adherence to the traditional "ACID" properties of transactions. More specifically, the present invention is directed at solving the "serializability" problems introduced by long lived transactions (LLT's). In addition to DBMS's and transaction processing monitors, the present invention may also be used in persistent programming languages as well as to concurrency control services for object resource brokerage systems. For simplicity, the present invention will be described with respect to DBMS's.

The idea of revising or redefining the ACID properties of transactions to enable more efficient execution of transactions in systems that support LLT's is not new. However, the present

invention provides a new methodology of "parameterized lock management" that is relatively simple to implement and that allows applications to explicitly control the degree to which they can tolerate diminished isolation between concurrent transactions.

The Traditional Transaction ACID Properties

Traditional database management systems (DBMS's) were developed to support short, atomic transactions, such as for banking and airline reservation applications. Standard transaction management uses flat transactions that adhere to the ACID properties. ACID stands for Atomicity, Consistency, Isolation and Durability.

Atomicity means that either the results of the transaction (i.e., changes to the database) are all properly reflected in the database, or none of them are. Generally, a transaction is said to commit or abort. When a transaction commits, all changes made to the database by the transaction are durably stored, leaving the database in a consistent state. When a transaction aborts, any changes made to the database by the transaction are backed out, once again leaving the database in a consistent state.

Consistency means that each transaction commits only legal results to the database. Thus, a transaction must bring the database from one consistent state to another consistent state.

Isolation means that the events within a transaction must be hidden from other transactions running concurrently. Concurrent transactions must not interfere with each other. They execute as if they had the database to themselves.

Durability means that once a transaction has been completed and has committed its results to the database, the system must guarantee that these results survive any subsequent malfunctions.

The concept of atomicity for transactions is sometimes overloaded with additional meaning. In particular, sometimes atomicity is defined to mean "concurrency atomicity," meaning that no transaction can observe any partial results of an atomic transaction. This document and the present invention, however, take the opposite approach. In particular, in this document atomicity is defined to mean that a transaction's commitment must be atomic. That is, once

some work is committed to the DBMS (as opposed to committed to the parent of a subtransaction), the transaction in question cannot continue to perform work that may or may not be committed at some later point in time. This notion of atomicity excludes such things as open nested transactions, but does not exclude partial rollbacks, the use of persistent savepoints, or other mechanisms that can be used by application programmers to control the behavior of the system's recovery mechanisms, since the final outcome of the transaction is still abort or commit.

Failure atomicity implies that all effects of incomplete transactions must be undone in the case of failure. Failure atomicity may be undesirable for long lasting transactions (LLT's). For example, a designer who experiences a power failure just as he is about to commit a week's worth of work is unlikely to consider failure atomicity to be a valuable property of the design database. The obvious solution for this situation (as well as many other LLT's) is for LLT's to have persistent savepoints, which would make it possible to recover an incomplete transaction to the last savepoint taken before a crash. It is noted that removing failure atomicity does not require removing commitment atomicity.

The atomicity and durability properties of transactions are required for system failure recovery, the isolation property is required for concurrency control, and the consistency property is needed for both failure recovery and concurrency control.

It is well known that the ACID properties are well suited for virtually all kinds of short duration transactions. LLT's seem to be the only class of transactions where the ACID properties cause significant problems. LLT's can be complex queries that last for minutes or hours, data mining queries that last for hours or days, or concurrent engineering transactions, controlled by humans and lasting from minutes to months. Full application of the ACID properties in a DBMS that supports LLT's can effectively prevent multiple users from simultaneously using the system. Long term locking of system resources by an LLT can prevent other users or transactions from being able to perform useful work.

It is a premise of the present invention that three of the ACID properties remain highly desirable for LLT's:

- Keeping commitment atomicity for LLT's is universally acknowledged as being desirable. Just like short duration transactions, LLT's should have only two possible outcomes: commit all work or abort all work (but see comments above regarding failure atomicity). Therefore, retaining commitment atomicity is desirable.
- Inconsistencies in a database are undesirable, although some kinds of inconsistencies may have to be tolerated when dealing with LLT's. Maintaining consistency is desirable, even for LLT's.
- Whether or not a transaction lasts a long time before it commits, durability for a committed transaction is always desirable.

It is a premise of the present invention that isolation is the only ACID property that it is desirable to compromise so as to reduce the impact of LLT's on system performance. More specifically, it would be desirable to compromise isolation in a controlled manner so as to give rise to as little inconsistency in the database as possible.

It is a primary object of the present invention to provide a lock manager mechanism for providing applications the ability to explicitly control the extent to which concurrent transactions are isolated from each other or share data with each other.

Working with Data That is Less Than 100% Reliable

There are situations in which some of the data values used by a transaction are missing or completely unknown. Missing information is sometimes called null data. The use of null data in a database management system (DBMS) requires extensions to the framework used for predicate evaluation during query execution. In particular, when a predicate is evaluated against a null, the result of the evaluation cannot be true or false. Thus, to allow or support evaluations that involve nulls, one must introduce a value called "maybe" or "unknown" as a truth value, or introduce Boolean nulls. Since predicates can be combined by means of the operators AND, OR and NOT to form arbitrarily complex predicates, rules are needed to specify the results of evaluating complex predicates. This is usually done by means of an appropriate form of algebra. For example, when there are no nulls involved, evaluation of complex predicates is performed according to Boolean algebra. Generally, prior art DBMS's

that support null data support only one kind of null data and evaluate complex predicates having one or more null data values using three valued logic.

The present invention was developed, in part, based on the premise that it would be desirable in many applications to have more than one kind of null, where the different types of nulls represent the different reasons why information is missing (e.g., "unknown," "divide by zero," "undefined," "prohibited," and so on). Although it is possible to allow multiple kinds of null data and still use three valued logic, using a three valued logic process to evaluate truth expressions having multiple kinds of null data effectively erases all information about the kinds of null data involved.

Furthermore, in addition to data being missing, there are situations in which data values are not missing, but are also not 100% reliable. For example, when a transaction uses "browse" access to read data, some of the data read by the transaction may be uncommitted data written by active transactions. More specifically, some of the data read by the transaction with browse access may be write locked by other transactions, while other portions of the data read may not be write locked by any other transactions. As a result, some of the data (i.e., the data not write locked by any other transaction) read by the transaction with browse access will be 100% reliable, while some other portions of the data (i.e., the data write locked by other transactions) will be less than 100% reliable.

When a write lock is held by a long lived transaction (LLT), the transaction is likely to have different phases and the values of the data on which write locks are held may have corresponding levels of reliability. Typically, data values for which an LLT hold write locks will be least reliable at the beginning phases of the LLT and more reliable at the later phases of the LLT. The prior art, however, treats all such data uniformly as either unreliable data or null data.

It is a goal of the present invention to provide a more flexible mechanism for evaluating queries that are functions of data having multiple levels or classifications of unreliability. In particular, it is a goal of the present invention to provide a query evaluation technique that classifies data with respect to its type, degree or classification of unreliability and that

maintains and properly combines data unreliability classification information as it evaluates complex query expressions.

SUMMARY OF THE INVENTION

In summary, the present invention is a database management system (DBMS) that has been modified to provide improved concurrent usage of database objects, particularly when the system is executing long lived transactions. A subset of the transactions access database objects using parameterized read and parameterized write access modes. Each transaction using a parameterized write mode of access for a database object specifies a write access mode and a write access mode parameter, where the write access mode parameter indicates a reliability classification that indicates the reliability of the write locked data. Each transaction using a parameterized read mode of access for a database object specifies a read access mode, and a read access mode parameter, where the read access mode parameter indicates one or more reliability classifications that are acceptable to the transaction. Whenever a transaction requests access to a database object, the DBMS generates a corresponding lock request for the object. If the access request is a parameterized conditional access request, a corresponding parameterized lock request is generated.

A lock manager processes each lock request by checking to see if any previously granted lock is conflicting or potentially conflicting with the requested lock. Two lock requests are unconditionally conflicting if their resource range overlaps and the access modes of the two requests are incompatible. Two requests are conditionally conflicting if analysis of their read/write parameters is necessary to determine whether a conflict exists. If the lock request being processed is unconditionally conflicting with any outstanding, previously granted lock, the lock request is put on a queue of pending requests. If the lock request is not unconditionally conflicting with any outstanding, previously granted locks, but is conditionally conflicting with an outstanding, previously granted lock, the conditional conflict is resolved by determining whether the write parameters for the write lock in question are a subset of the read parameters for the read lock in question. If so, then there is no conflict. If not, then the requested lock is in conflict with the outstanding previously granted lock. In none of the outstanding, previously granted locks is in conflict with the requested lock, the requested lock is granted. Otherwise it is put on a queue of pending lock requests.

Every time a previously granted lock is released, any pending lock requests that overlap with the resource associated with the released lock are reevaluated.

Another aspect of the present invention is a data processing method suitable for use in a database management system (DBMS) or other data processing system or product that stores and processes information. A subset of the stored information is stored as annotated values, each annotated value having a stored data value and an associated data reliability value. That is, the data value is annotated with a data reliability value. The data reliability value associated with each datum is a member of a domain that includes at least three distinct data reliability values corresponding to at least three distinct data reliability classifications. Typically, one of the data reliability classifications represents fully reliable data and at least two of the distinct data reliability classifications represent two distinct classifications of less than fully reliable data. For example, the domain of reliability categories might include the following six reliability categories: fully reliable, good, medium, poor, unknown and missing.

A query executer or other data processing procedure executes queries (or other data processing tasks) requesting access to information stored in the database. The query executer includes annotated nullable logic for evaluating expressions containing at least one annotated value as an operand. The annotated nullable logic includes logic (i.e., instructions) for combining annotated values, for comparing annotated values so as to generate annotated truth values, and for combining annotated truth values in accordance with a predefined set of rules. The annotated nullable logic generates result values in a manner that preserves the relevant data reliability values associated with the annotated data and/or truth values that have been combined and/or compared.

In a preferred embodiment, the data reliability values associated with the annotated values are each encoded as a reliability category bitmap having a multiplicity of bits. Each bit of the bitmap corresponds to a respective data reliability classification. Since fully reliable data does not need to be annotated, or can be annotated with a bitmap whose bits are all set to zero, the number of bits in the annotation bitmap is typically one less than the number of distinct data reliability classifications.

The annotated nullable logic includes means for combining first and second annotated values (k, A) and (m, B) using a predefined operator Op to generate an annotated result value (r, C):

$$(k, A)\ Op\ (m, B) = (r, C)$$

where k, m and r represent data values or truth values, and A, B and C represent data reliability values associated with the annotated first, second and result values, respectively. For some types of operations, such as numeric combining operations (addition, subtraction, multiplication, division, numeric value comparison), character manipulation operations and certain logic operations (e.g., AND, NAND, XOR, XNOR), the annotation bitmap for data reliability value C is generated by forming a bitwise union of the bitmaps for operands A and B. For other types of operations, such as OR'ing logical operands, the bitmap for data reliability value C is generated by selecting one operand's annotation bitmap (e.g., in the case of the OR operation, the annotation bitmap for operand having the strongest truth value is selected).

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

Fig. 1 is a block diagram of a database management system implementing the present invention.

Fig. 2 depicts a set of lock management data structures for keeping track of granted and pending resource lock requests.

Fig. 3 depicts an alternate set of lock management data structures for keeping track of granted and pending resource lock requests.

Fig. 4 is a flow chart of the object locking method of the present invention.

Fig. 5A depicts the data structure of a meta-datum and its schema, and Fig. 5B depicts some examples of meta-data values.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to Fig. 1, there is shown a database management system (DBMS) 100. Transaction operation and management are handled by a transaction manager 102 and a lock manager 104, both of which are software procedures executed by the system's data processor(s) 106. The transaction manager maintains a transaction table 108, sometimes implemented as a tree structure, for keeping track of the identity and status of all pending transactions. The lock manager 104 maintains a lock table 110, usually implemented using a hash table and various linked lists and/or tree structures (which will be discussed in more detail below). The lock table 110 keeps track of all locks that have been requested on resources in a database 112. The database 112 stores all of the data that is accessible to transactions executed by the DBMS 100.

The DBMS 100 will typically also include additional memory resources 114, one or more system busses 116 for interconnecting the various parts of the system, and a network interface 118 or other communications interface for handling communications with client workstations 120.

"Conflicting transactions" are two or more transactions that access the same resource and at least one of them will, at least potentially, update the resource in question. Thus the results generated by at least one of the conflicting transactions may depend on the order in which the transactions are performed.

A "data lock" is a mechanism for assigning a transaction certain rights to a database object, such as a table, a page, or a datum or record in a database table. Thus a first transaction may lock a particular object so as to ensure that no other transaction accesses the data in that data until the first transaction commits or aborts. The prior art includes many types of data locking mechanisms.

"Overlapping resources" are database objects whose address range is the same or at least partially overlapping.

An "access mode" refers to the way in which a transaction or application accesses a data resource. The traditional access modes (browse, read, update, write and exclusive) are described below. When using the parameterized read and write access modes of the present invention, each unique read access parameter value is associated with a distinct read access mode, and each unique write access parameter value is associated with a distinct write access mode.

Lock Table Data Structures Supporting Parameterized Resource Access Requests and Access Modes

Referring to Fig. 2, the "lock table" 110 in a preferred embodiment is implemented as follows. A hash function 150 is used to convert a resource identifier into a hash table address in a fixed size hash table 152. The resource identifier that is hashed by function 150 may include a resource type or level indicator (e.g., indicating whether the resource to be locked is a database, table, page or tuple) and the starting address of the resource. Each addressable slot 154 of the hash table includes either a null value if there are no locked resources corresponding to that slot's hash table address, or a pointer to a list of lock control blocks (LCB's) 160 if there is at least one locked resource whose hash value corresponds to that slot's hash table address.

The lock manager will allocate (i.e., generate and store) one lock control block (LCB) 160 for each lockable data item that is actually locked, and will allocate (i.e., generate and store) one lock request block (LRB) 162 for each lock held by a transaction. Thus, if a particular database object is locked by three different transactions at a given point in time, there will be one LCB 160 for that object and a linked list of three LRB's (one per transaction) "hanging" from that LCB.

Each LCB 160 preferably includes:

- a lock ID 170, which will typically include a copy of the resource identifier for a locked resource;

- a mode indicator 171 indicating the most restrictive access mode (e.g., browse, read, parameterized read, write, parameterized write, or exclusive) of all the locks granted on the database resource represented by this LCB;
- a read parameters indicator 172, preferably in the form of a bitmap, representing the logical OR of the read parameters being used by the parameterized read locks (if any) outstanding on the locked resource;
- a write parameters indicator 173, preferably in the form of a bitmap, representing the write parameters of the parameterized write lock (if any) outstanding on the locked resource;
- a granted request list pointer 174 to a list of LRB's 162 for granted (i.e., currently outstanding) resource requests for the database resource represented by this LCB;
- a pending request list pointer 175 to a list (also called a queue) of LRB's 162 for pending (i.e., not yet granted) resource requests represented by this LCB; and
- a next LCB pointer 176 to the next LCB (if any) sharing the same hash address as this LCB.

The read and write parameters represented by fields 172 and 173 in LCB 160 represent an extension by the present invention to the conventional access modes used by DBMS's, and are discussed in more detail below. Each distinct value of a defined set of read/write parameter domain represents a corresponding data reliability classification or category. Thus a parameter domain having eight parameter values (each represented by a corresponding bit of an eight-bit parameter field) would allow for the definition of up to eight different data reliability classifications. In other embodiments of the present invention the parameters may be used to indicate properties of a database object other than "reliability," such as the type or identity of the application which holds a write lock on the object, or other information that can be used by applications to determine if they are willing to read the data despite the presence of a write lock on it.

Each LRB 162, representing one granted or pending lock request, preferably includes:

- a mode indicator 181, indicating the access mode (e.g., browse, read, parameterized read, write, parameterized write, or exclusive) in which the resource is being accessed or being requested by a particular transaction;
- a transaction identifier 184, which identifies the transaction that requested or holds the lock corresponding to this LRB;

- a parameters indicator 182, preferably in the form of a bit map, representing the read or write access mode parameters (if any) being used by the owner of this read or write lock; this field is used only if the owner of this lock or lock requested is using a parameterized access request;
- a lockset pointer 185 that is used to form a linked list of all the LRB's owned by transaction identified by the transaction ID 184; and
- a Next LRB pointer 186 to the next LRB (if any) for the same database resource as this LRB.

Typical sizes for the read and write parameter fields in the LCB's and the access mode parameter field in the LRB's are one to two bytes, supporting eight to sixteen distinct parameter values.

Fig. 3 depicts an alternate data structure implementation 110' of the lock table. In this implementation, the lock control blocks (LCB's) 190 each contains a granted request bitmap 194 and a pending request bitmap 195 in place of the granted and pending request list headers 174 and 175. Each of the request bitmaps preferably contains a bitmap of sufficient size (e.g., 64 or 128 bits) to represent the maximum number of concurrent transactions supported by the system. Every active transaction is assigned to one of the bitmapped transaction identifiers. The granted request bitmap 194 contains a set bit for each active transaction that has been granted a lock on the resource represented by the LCB 190. Similarly, the pending request bitmap 195 contains a set bit for each active transaction that has a pending lock request (also called an access request) on the resource represented by the LCB 190.

The read and write parameter bitmaps 172 and 173, as in the Fig. 2 implementation, represent all the read and write parameterized access modes that have been granted on the database resource represented by the LCB 190.

Whenever a lock request is granted, the read and write parameters in the LCB in question are efficiently updated by logically ANDing the read/write parameters of the granted lock request with the read/write parameter bitmaps previously stored in the LCB. Ideally, when a lock is released, one would like the read and write parameters in the corresponding LCB to be immediately updated. However, this would require ORing the read parameters of all remaining

read lock holders (if a read lock is released) and ORing the write parameters of all remaining write lock holders (if a write lock is release). Since this would put an undue delay on transaction commit processing, in a preferred embodiment the lock is released without updating the read and write parameter bitmaps in the LCB. If there are no more lock holders for the resource in question, the read and write parameter bit maps can be cleared, and if there are no pending lock requests, then the LCB can be eliminated altogether. Otherwise, the read and write parameter bitmaps in the LCB are updated with respect to lock releases only when a potential read/write or write/read conflict is detected, in order to figure out whether to grant the requested lock. Alternately, LCB read/write parameter bitmap updating can be handled as a background task, similar to garbage collection.

Serializable Transaction Histories

A transaction history is defined to be serializable if it is equivalent to a serial transaction history. This definition only makes sense if we also define what it means for two histories to be equivalent and what it means for a history to be serial. Equivalence can be defined in more than one way. Two histories are "conflict equivalent" if:

- they contain the same transactions and the same operations; and
- conflicting operations of non-aborted transactions are ordered the same way in both histories.

Two operations are defined to conflict if they do not commute. That is, if the results of executing one before the other is in general not equivalent to executing them in the reverse order. It is common to model transactions as consisting of read and write operations only, and the conflict relation for those two operations is that two operations conflict if at least one of them is a write operation. The basic lock compatibility matrix is:

	R	W
R	*	
W		

Table 1

where "*" in Table 1 (as well as in the following tables) indicates non-conflicting operations.

Table 1 indicates that two transactions can read the same data item, but a transaction that is performing write operations must have exclusive access to it.

A transaction history H is serial if for every pair of transactions T_i and T_j, either all operations of T_i come before all operations of T_j in H, or all operations of T_j come before all operations of T_i in H. In other words, a transaction history is serial if it does not have any concurrency. Each transaction executes to completion before the next one starts. If transactions are atomic, durable and consistent, then a serial transaction history will be correct. It follows that a concurrent execution of transactions that is conflict equivalent to a serial one, must necessarily be correct, too. A transaction history that is conflict equivalent to a serial history is called *conflict serializable*, and the corresponding correctness criterion is called *conflict serializability*.

A serialization graph (SG) for a transaction history H is denoted SG(H). This is a directed graph whose nodes are the committed transactions of H, and it has an edge between all pairs of nodes representing transactions that have issued conflicting operations. The direction of the edges are in accordance with the sequence of the conflicting operations. An edge from T_i to T_j in SG(H) means that T_i as issued an operation that conflicts with and precedes some operation issued by T_j. Intuitively, if T_i and T_j are involved in a cycle in SG(H), then T_i comes both before and after T_j in H, in which case H cannot be equivalent with any serial history. The fundamental theorem of serializability says that a history H is serializable if and only if its serialization graph SG(H) is acyclic.

To enforce serializability, virtually all commercial DBMS's use some form of data locking. Two phase locking (2PL) operates according to the following rules:

1) a transaction may not perform an operation on a data item unless it holds a lock corresponding to the operation (e.g., a read or write operation) in question on that data item;

2) a lock request from a transaction must be delayed or rejected by the transaction scheduler if another transaction holds a conflicting lock on the data item in question; and

3) a transaction may not acquire a new lock if it has released any of its old ones.

The first two rules prevent transactions from directly interfering with each other. The third rule, called the two phase locking rule, prevents cycles in the serialization graph. The intuitive explanation of the two phase locking rule is as follows. When a transaction acquires a lock, that may establish an incoming edge to its node in the serialization graph. An outgoing edge from a transaction's node in the serialization graph can only be established if that transaction has released a lock. Thus, in order to create a cycle in the serialization graph, some transaction must first release a lock and then later acquire a lock. Since this is prohibited by the two phase locking rule, transaction schedulers that obey the two phase locking rule ensure acyclicity of serialization graph's, and therefore ensure serializable histories.

In addition to serializability, it is important for a transaction management system to maintain a "strict" history by:

- avoiding cascading aborts (ACA), in which the failure of a first transaction causes other transactions that depended on the results computed by the first transaction to abort; and
- ensuring that all data items written by a transaction T_i cannot be overwritten by another transaction until after transaction T_i has aborted or committed.

The solution for maintaining a strict transaction history is to add the restriction that all write locks acquired by a transaction must be held until the transaction commits. A rigorous transaction history is maintained by requiring that all read and write locks acquired by a transaction be held until the transaction commits. With these added restrictions, two phase locking is called "strict two phase locking," or strict 2PL.

A nasty problem with transaction schedulers that use the two phase locking rule is that transactions can get involved in deadlocks. As a consequence of the second locking rule, transactions sometimes have to wait for locks. Such waiting is caused by another transaction holding a conflicting lock, and the waiting transaction cannot make any progress until the other transaction releases its lock. If two transactions are waiting for each other, neither can make progress until the other one releases its lock. As long as neither of them releases its lock, the two transactions are deadlocked. More generally, deadlocks can involve more than two transactions that are waiting for each other in a cyclic way.

Transaction Isolation Levels

While isolation levels (i.e., levels of isolation between transactions) can be defined without reference to any particular implementation technique, the present invention assumes the use of a lock manager for implementing isolation control. Isolation levels supported by commercial systems, ordered from least to most restrictive, include:

- UR (uncommitted read). This is also known as read uncommitted, read through, or dirty read. This isolation level allows an application to read both committed and uncommitted data. As a result, the data read by an application (i.e., transaction) may be inconsistent with other data read by the application. Applications using the UR isolation level do not acquire any ordinary read locks, but will typically hold a browse lock at some high level in the resource hierarchy. Applications using the UR isolation level must still use write locks to protect write operations. There are two types of situations where UR isolation is most typically used: (1) when the application does not need an exact answer, and (2) when the application (i.e., the person who wrote the application) knows that the data to be read is not being updated by anyone else.
- CR (committed read). This is also known as read committed. The committed read isolation level allows an application to read only committed data. CR isolation can be implemented using "zero duration" read locks. That is, if an application wants to read a database object, if suffices for the DBMS to check that a read lock could have been granted. If the read lock could have been granted, the object is read, but a read lock is not acquired. CR isolation is much less expensive than cursor stability (CS) isolation (described below) in terms of CPU cycles. Unless an application needs the additional semantics offered by cursor stability isolation (and many applications do not), committed read isolation is the better alternative.
- CS (cursor stability). Cursor stability isolation allows an application to read only committed data and guarantees that a row will not change as long as a cursor is positioned to it. CS isolation causes locks to be kept until the cursor moves to the next lockable object. For example, CS isolation is useful for an application that fetches a row from a database table (using a cursor to point to the row) and then performs a database manipulation based on the current row's data values before fetching another row. An application should have at least CS isolation for cursors that are to be used to point to rows of data that are to be updated or deleted by UPDATE or DELETE operations.

- RR (repeatable read). RR isolation allows an application to read only committed data and guarantees that read data will not change until the transaction terminates (i.e., a read that is repeated will return the original row, unchanged). RR isolation will not prevent the so-called phantom row phenomenon. That is, when a cursor is reopened, a row not present the previous time may appear. Read locks covering data items retrieved by an application using RR isolation must be kept until the transaction commits or aborts.
- TC (transaction consistency). TC isolation is also known as serializable isolation. TC isolation allows an application to read only committed data and guarantees that the transaction has a consistent view of the database, as if no other transactions were active. Transactions using TC isolation may be part of a transaction history that is not serializable where other transactions use lower levels of isolation. On the other hand, if all transactions in a history use TC isolation, that history will be serializable. All read locks acquired by a transaction using TC isolation must be kept until the transaction commits or aborts.

Another isolation level is the QC (query consistency) level. QC isolation allows an application to read only committed data, and guarantees that all data accessed in a single query is consistent. Implementing QC isolation by means of data locking is straightforward: all read locks must be kept until the query is completed (i.e., until the cursor is closed). Since cursors are defined using queries, QC isolation guarantees that all rows in a cursor's answer set are consistent. QC isolation is also valuable when performing statistical analysis, or when comparing or otherwise using together data values from different cursor row occurrences. QC isolation is weaker than RR isolation, but stronger than CS isolation.

Isolation level CS, or weaker, is often not suitable for any query that uses aggregation and for any application that processes a cursor where the answer set must be consistent. While using RR or TC isolation solves this problem, RR and TC actually provide more isolation than is needed for most such applications. The QC isolation level is the lowest isolation that provides the necessary and sufficient isolation for such queries.

Transaction Access Modes

There are two basic access modes: read and write. When a database object is accessed in read mode, the agent in question can perform only read operations on that object. Two or more transactions may access a given object concurrently, provided they all use read mode. When a database object is accessed in write mode, the transaction in question can perform both read and write operations on that object. More specifically, write mode enables reading, deleting, inserting and modifying objects. If an object is accessed in write mode by one transaction, no other transaction can access that object in either read or write mode. In addition to these two basic access modes, many DBMS's support browse, upgrade and exclusive access modes.

Browse mode enables a transaction to read an object even if some other transaction is currently accessing it in write mode. Thus, when using browse mode, transactions have no guarantee of a consistent view of the database, since there is a risk that they will read uncommitted data. The use of browse mode is often denoted as read through mode or dirty read mode, and is used with isolation level UR. Even an application using the UR isolation level needs to inform others of its presence, and it does so by accessing resources in browse mode.

Upgrade mode is similar to read mode, with the added semantics that the transaction in question may at any time request an upgrade to write mode. That is, it may upgrade its access mode. When a first transaction accesses an object in upgrade mode, no other transaction can access the same object in write mode, or upgrade mode, until the first transaction commits or aborts.

Support for upgrade mode was added to DBMS's to prevent single object deadlocks. Some applications work as follows: a number of database objects are "looked at," but only some of these are updated or deleted. If all the objects in question are "looked at," in write mode, the problem is unacceptably low concurrency. The alternative, assuming upgrade mode is not supported, is to "look at" objects in read mode and then promote from read to write mode whenever an update or delete operation is to be performed. The problem with this approach is that two transactions may access the same object in read mode, and if they both request promotion to write mode, the result is deadlock. This dilemma is eliminated by supporting upgrade mode, since upgrade modes are not mutually compatible.

Exclusive mode is used by a transaction to prohibit any other transactions from accessing the same object, irrespective of access mode. Exclusive mode is used when even browsers must be denied access to an object, such as when a table is to removed from a relational database.

The relationship between the five traditional access modes is represented by Table 2, where B represents browse mode, R represents read mode, U represents upgrade mode, W represents write mode and X represents exclusive mode.

	B	R	U	W	X
B	*	*	*	*	
R	*	*	*		
U	*	*			
W	*				
X					

Table 2

Asterisks (*) in Table 2 indicate compatibility. For example, read mode (R) access by a first transaction is compatible with browse (B), read (R) or upgrade (U) mode in a second transaction. Write mode (W) is compatible only with browse mode. Exclusive mode (X) is not compatible with any other access mode.

The five access modes are typically implemented using locks. It should be noted that it is usually necessary to support a number of different lock granularities. Typical lock granularities are tuple (i.e., database table row), object, page, table, class, file, tablespace, and database. Unless it is required that all transactions use the same lock granularity, the DBMS must be able to coordinate concurrent transactions that request locks at different levels in the resource hierarchy. The typical solution is once a transaction requests locks at some resource level, it will not request additional locks on lower levels of the same resource (because other concurrent transactions may acquire other locks on portions of that resource that are compatible with the first lock put on the resource), but it may request locks at higher levels. It is possible for a single transaction to use different lock granularities for different statements, but this is not significant for the discussion at hand.

The following intent locks are needed: intent to request read (IR) locks indicate an intent to request read locks at some lower level, intent to request write (IW) locks indicate an intent to request write locks at some lower level, and RIW, which provides read access to the entire resource in question (e.g., a table) while also enabling the transaction to request update and write locks at some lower level (such as at the page or tuple level). It should be noted that an IW lock on a table enables its holder to request R, U and W locks (not just W locks) on tuples or pages with the table. An IW lock on a resource will be promoted to an RIW lock if the transaction holding the IW lock requests an R lock on the same resource.

Two transactions with overlapping IW locks are considered to be compatible because the potential conflicts will be resolved at some lower level in the resource hierarchy. For example, two transactions may need to update various tuples in a relational table. The both acquire IW locks on that table (and probably also on some higher level resources, such as file or tablespace, as well as the database), and then R, U or W locks on individual tuples. As long as the two transactions do not access the same tuple, then there will no conflict. Should they happen to access the same tuple, then there will be a conflict, and the possibility of deadlock cannot be completely excluded. However, the potential deadlocks caused by overlapping IW requests are, in general, no worse than the potential deadlocks associated with other resource locking situations.

The complete lock (access mode) compatibility matrix is shown in Table 3.

	B	IR	R	U	IW	RIW	W	X
B	*	*	*	*	*	*	*	
IR	*	*	*	*	*	*		
R	*	*	*	*				
U	*	*	*					
IW	*	*			*			
RIW	*	*						
W	*							
X								

Table 3

Compatibility Matrix for Access Modes

Note that the RIW row/column is identical to the intersection of the R and IW rows/columns. In practice, browse (B) locks and exclusive (X) locks are not used at the lowest resource levels. Using B locks at the lowest levels would, at least partially, defeat the purpose of using isolation level UR in the first place. For example, in a relational DBMS a transaction using the UR isolation level may request a browse lock at the table level (and all levels above the table level), and then proceed without requesting any locks on pages or tuples. Alternately, the transaction may request some sort of low-cost, short duration locks (known as latches) on pages or tuples to ensure atomicity of individual read operations.

Conditional Conflict Serializability

The present invention is based on the premise that serializability is an unsuitable correctness criterion for some types of applications, such as concurrent engineering, typified by CAD and CASE applications. The present invention uses a new correctness criterion, herein called conditional conflict serializability (CCSR) which is a weaker kind of serializability based on a weaker notion of conflict between transactions.

The idea behind CCSR is to depart from a purely commutativity based definition of conflict. While write operations are still considered to be mutually conflicting, write-read and read-write

conflicts are made conditional. This is achieved by using "parameterized" read and write modes, and corresponding parameterized read and write locks.

If R(A) and W(B) denote parameterized read and write modes, respectively, where A and B denote subsets of some parameter domain D, R(A) and W(B) are compatible if and only if B is a subset of A: $B \subseteq A$. For example, if the parameter domain D contains modes u1, u2, through u5, R(u1, u2) is compatible with W(u1) because $u1 \subseteq (u1, u2)$.

Recall that two transaction histories are "conflict equivalent" if they contain the same transactions and operations, and conflicting operations of non-aborted transactions are ordered in the same way in both histories. Identical terms can be used to defined conditional conflict equivalence, so long as the parameter subset comparison is used to determine which operations are conflicting. Thus, a transaction history is defined to be "conditional conflict serializable" if and only if it is conditional conflict equivalent to a serial transaction history.

As discussed above, the serializability theorem states that a history H is serializable if and only if its serialization graph is acyclic. A generalized version of this theorem applies to CCSR. A conditional conflict serialization graph (CCSG) is defined in the same way as a regular serialization graph, provided the term "conflicting operations" is understood to mean conditionally conflicting operations. Thus, a transaction history H is conditional conflict serializable (CCSR) if and only if the history's conditional conflict serialization graph (CCSG) is acyclic.

The use of dirty reads does not provide a means for distinguishing between relatively stable database data and database data undergoing major changes. All sorts of inconsistencies can result from dirty reads, and traditional DBMS's do not provide the user with any hints as to what they might be. On the other hand, the parameterized access modes of the present invention make it possible for database users (i.e., typically, application programs) to receive information about the reliability of uncommitted data from the parameters the writer of the data has attached to the write lock on the data. More generally, the parameterized access modes of the present invention (A) enables application programmers to customize the notion of conflict between transactions, and (B) enables applications to communicate to each other the quality of uncommitted data.

The parameter domain D can be user defined, or defined differently in different database systems. The "data model" used can be simple or complex, and thus the number of parameters in domain D can be small or large, depending on the needs of the application programs.

Using the present invention, the standard assumption that read and write modes are mutually incompatible is reduced to a default, which transactions can override by proper use of parameters. Instead, applications or transactions can specify when reading and writing is incompatible. Using the example discussed above in which the parameter domain D contains modes u1, u2, through u5:

R(u1, u2) and W(u1) are compatible,
R(u1) and W(u1) are compatible,
R(u1) and W(u2) are not compatible, and
R(u2) and W(u2, u3) are not compatible.

Non-parameterized read and write modes can still be denoted as R and W, but can be thought of as R(Ø) and W(*), where Ø denotes the empty set and * denotes an arbitrary superset of D. Thus, according to the rule that R(A) and W(B) are compatible if and only if $B \subset A$, R(Ø) is incompatible with all write modes and W(*) is incompatible with all read modes. Generally, there will be no such thing as a write mode that is compatible with every read mode, but R(D) denotes the read mode that is compatible with every write mode W(B) except W.

When an application uses a parameterized write mode, it is indicating a willingness to share information with readers. That is, a transaction using parameterized writes indicates to other transactions the degree of safety associated with reading data that it has not yet committed. Analogously, the use of parameterized read modes by a transaction indicates willingness to read data that belongs to parameterized writers. The new access mode compatibility matrix, using parameterized access modes, is shown in Table 4. In Table 4, * indicates unconditional compatibility, blank table entries indicate unconditional incompatibility, and formulas such as $B2 \subset A1$ and $B1 \subset A2$ indicate conditional compatibility.

	B	IR(A1)	R(A1)	U	IW(B1)	R(A1)IW(B1)	W(B1)	X
B	*	*	*	*	*	*	*	
IR(A2)	*	*	*	*	*	*	B1⊂A2	
R(A2)	*	*	*	*	B1⊂A2	B1⊂A2	B1⊂A2	
U	*	*	*					
IW(B2)	*	*	B2⊂A1		*	B2⊂A1		
R(A2) IW(B2)	*	*	B2⊂A1		B1⊂A2	B1⊂A2 AND B2⊂A1		
W(B2)	*	B2⊂A1	B2⊂A1					
X								

Table 4

Conditional Compatibility Matrix for Parameterized Access Modes

Parameterized lock modes increase the amount of memory used by the lock manager to record each lock, and also increase the amount of computation required to resolve lock requests.

It may be noted that the present invention does not force users (i.e., applications) to quantify uncertainty, but rather allows them to classify it. That is, users can denote unreliable data as belonging to one or more of a predefined set of reliability categories. Each such category is represented by a reliability classification indicator, that is by a parameter in domain D.

Especially for long lived transactions, it may be necessary to allow transactions to update the parameters associated with their write locks over time. For instance, the write locked data may initially be very unreliable (denoted by parameter u1), and then may be progressively upgraded to higher and higher levels of reliability (denoted by parameters, u2, u3, and so on) as the transaction progresses.

Further, in systems having long lived transactions, there may be a need for lock management methods that do not simply block a transaction when incompatible lock modes are encountered. Rather, in at least some situations, parameterized read modes may be treated as a request for data filtering, such that objects that are locked in incompatible write modes are skipped by parameterized reads. In some implementations of the present invention,

applications using parameterized read access requests may include an additional filter/wait flag to indicate if the read request is to be serviced by filtering out objects locked in incompatible write modes or is to wait for them to become available.

The "uncommitted dependency problem," the "dirty read" problem and the "temporary update problem" are three names for the same thing: the use of uncommitted data is unreliable because it is subject to further change, due either to transaction abort or further modification by the application. The real problem is that an application can retrieve uncommitted data "assuming" that it is reliable, and then go ahead and do something based on this assumption. Contrary to this, the parameterized access mode method of the present invention explicitly informs applications when uncommitted data is encountered and also delivers information about the reliability of the uncommitted data. The application is free to handle such a situation in whatever way it sees fit: it may continuing as if nothing special happened, it may invoke some special procedure, it may perform a full or partial rollback, or do something else. Since no application will be deceived into believing that uncommitted data can be fully trusted, the uncommitted dependency problem is eliminated when using the present invention.

The incorrect summary problem and the inconsistent analysis problem are two versions of another problem: an application may retrieve multiple data items and use them as input to some calculation, such as finding a sum or average, but interference from a concurrent transaction may cause the calculated value to be incorrect. This problem occurs when an updating application involves at least two data items: one that has already been retrieved by the analyzing application and one that has not yet been retrieved. One possible solution to this problem is to use an unparameterized read, which is incompatible with all write modes. However, this solution is undesirable or impossible in some situations.

Another possible solution, applicable only to special cases, is to establish a rule that all updates involving more than one data item should be performed using write mode parameters from a subset S of D. Any reader than needs to perform a consistent data analysis needs to avoid reading data that is write locked by any other transaction using a write mode parameter in subset S.

Another possible solution related to access mode usage is to label data modification as minor, medium and major (or any other set of gradations). Thus a user would have objects locked in W(minor) mode most of the time, but would upgrade to W(medium) or W(major) whenever more significant changes than usual are to be performed, such as shuffling the sequence of data items, deleting and inserting multiple data items, and so on. After the major changes have been made, the user would downgrade the write locks to W(minor). In this way, readers can protect themselves from reading data in the midst of undergoing major changes, while accepting smaller levels of data inconsistency.

Handling Resource Access Requests

Referring to Fig. 4, it can be assumed that in any system utilizing the present invention, a subset of the transactions access database objects using parameterized read and parameterized write access modes, while other transactions used unparameterized access requests. Each transaction using a parameterized write mode of access for a database object specifies a write access mode, and a write access mode parameter, where the write access mode parameter indicates a reliability classification to other transactions that may request read access to the database object. Each transaction using a parameterized read mode of access for a database object specifies a read access mode, and a read access mode parameter, where the read access mode parameter indicates one or more reliability classifications that are acceptable to the transaction. Whenever a transaction requests access to a specified database object, the DBMS generates a corresponding lock request for the object (step 220). If the access request is a parameterized conditional access request, a corresponding parameterized lock request is generated.

The system's lock manager processes each lock request by searching the lock table (steps 222, 224) for any corresponding previously generated locks. Next, it checks to see if any previously granted lock is conflicting or potentially conflicting with the requested lock (steps 226, 230, 232). Two lock requests are unconditionally conflicting if their resource ranges overlap and the access modes of the two requests are incompatible (step 226). For example, the blank positions in Table 4 represent unconditionally conflicting access requests. In terms of the data structures shown in Figs. 2 and 3, the access mode of the current request is compared with the most restrictive access mode previously granted for each of the overlapping resources for

which any locks have been granted. For example, if the current access request is for an upgrade (U) mode or a write (W) mode, and there is already a granted lock for an overlapping resource that has a U or W mode, the current access request is unconditionally conflicting with the previously granted lock. If the lock request being processed is unconditionally conflicting with any outstanding, previously granted lock (step 226), the lock request is put on a queue of pending requests (step 228).

Two requests are conditionally conflicting if analysis of their read/write parameters is necessary to determine whether a conflict exists (step 230). Using the access modes described above, the read/write parameter comparison performed for each pair of potentially conflicting requests are those shown in Table 4 (step 232). That is, the conditional conflict is resolved by determining whether the write parameters for the write lock in question are a subset of the read parameters for the read lock in question. If so, then there is no conflict. If not, then the requested lock is in conflict with the outstanding previously granted lock. If none of the outstanding, previously granted locks is in conflict with the requested lock, the requested lock is granted (step 234). Otherwise it is put on a queue of pending lock requests (step 228).

Every time a previously granted lock is released, any pending lock requests that overlap with the resource associated with the released lock are reevaluated (step 238).

Alternate Embodiments Of CCSR Lock Manager

In an alternate embodiment, the definitions of compatible read and write modes in CCSR can be generalized. In particular, the following are compatibility definitions for five alternate embodiments:

1) the R(A) and W(B) read and write modes are defined to be compatible if and only if A is a subset of B;
2) the R(A) and W(B) read and write modes are defined to be compatible if and only if A is equal to B;
3) the R(A) and W(B) read and write modes are defined to be compatible if and only if the intersection of A and B is empty;
4) the R(A) and W(B) read and write modes are defined to be compatible if and only if the intersection of A and B is non-empty; or

5) the R(A) and W(B) read and write modes are defined to be compatible if and only if f(A,B) yields a True result, where f(A,B) is any suitably defined truth-valued function.

The access mode compatibility matrix for alternate embodiment number 5, which is actually a generalized representation of all the embodiments, is shown in Table 5. In Table 5, * indicates unconditional compatibility, blank table entries indicate unconditional incompatibility, and formulas such as f(B2,A1) and f(B1,A2) indicate conditional compatibility (i.e., the modes are compatible if the function f() evaluates to True).

	B	IR(A1)	R(A1)	U	IW(B1)	R(A1)IW(B1)	W(B1)	X
B	*	*	*	*	*	*	*	
IR(A2)	*	*	*	*	*	*	f(B1,A2)	
R(A2)	*	*	*	*	f(B1,A2)	f(B1,A2)	f(B1,A2)	
U	*	*	*					
IW(B2)	*	*	f(B2,A1)		*	f(B2,A1)		
R(A2) IW(B2)	*	*	f(B2,A1)		f(B1,A2)	f(B1,A2) AND f(B2,A1)		
W(B2)	*	f(B2,A1)	f(B2,A1)					
X								

Table 5

Conditional Compatibility Matrix for Parameterized Access Modes

As stated above, parameterized lock modes increase the amount of memory used by the lock manager to record each lock, and also increase the amount of computation required to resolve lock requests.

COMBINING META-DATA OF VARYING DEGREES OF RELIABILITY

A "data lock" is a mechanism for assigning a transaction certain rights to a database object, such as a table, a page, or a datum or record in a database table. Thus a first transaction may lock a particular object so as to ensure that no other transaction accesses the data in that data

until the first transaction commits or aborts. The prior art includes many types of data locking mechanisms.

An "access mode" refers to the way in which a transaction or application accesses a data resource. The traditional access modes are browse, read, update, write and exclusive. Other access modes include parameterized read and write access modes. These are described above. For the purposes of this document, it is sufficient to know that when a transaction accesses data using a parameterized write access mode, the value of the associated write access parameter may be suitable for interpretation as a data reliability classification or category for the data values covered by the write lock held on that data. If the DBMS provides a read/write parameter domain having eight parameter values (each represented by a corresponding bit of an eight-bit parameter field), that system would allow for the definition of up to eight different data reliability classifications for data covered by parameterized write locks (i.e., there might be additional data reliability classifications for data not covered by parameterized write locks).

As indicated earlier, for a variety of reasons and within many different application domains, data values in a database are not always accurate. While it is often difficult, if not impossible, to *quantify* the degree of uncertainty for a given piece of information, it is much easier to *classify* the information in question. For example, information could be classified as being:

- based on samples;
- based on polls;
- based on extrapolation;
- based on interpolation;
- based on estimates;
- based on guesstimates;
- based on rumors;
- based on presumably biased reports;
- based on neutral sources;
- based on statistics of reliability class x;
- measured with an instrument of reliability class y;
- likely to change in the (near) future;
- continuously changing; or
- a snapshot.

Many other classifications could be useful, depending on the application domain.

Meta-Data

Unlike prior art three-valued logic systems that utilize a special "unknown" value, the present invention annotates both data and logical values that are less than fully reliable. Instead of replacing these values with a special "unknown value," they are annotated with a meta-value representing a reliability classification.

Referring to Figs. 5A and 5B, information or data that has been assigned a reliability classification will be called meta-data or meta-information. Meta-data provides information about the data that is being manipulated. Fig. 5A represents one possible schema for representing the reliability of a meta-data value. In this example a meta-datum 350 is encoded as a data value 352 and a meta-value 354. The schema 360 for the meta-datum, which would typically be stored as part of the schema for a database table or object, includes: a data type 362 for the data value 352, and a meta-value type 364 specified using a bitmap representation.

In this simplified example the meta-value bitmap has five defined bit fields. The first two bits of the meta-value are used to indicate cases of missing data, while the third through fifth bits are used to indicate degrees of reliability of data that is not missing. More specifically, the first bit is set when a datum's value is not applicable, the second bit is set when a datum's value is unknown, the third bit is set when the reliability of a datum's value is poor, the fourth bit is set when the reliability of a datum's value is medium, and the fifth bit is set when the reliability of a datum's value is good. Fig. 5B represents some examples of meta-data values. Note that a data value that is 100% reliable is indicated by a zero meta-value.

More generally, the meta-value associated with a meta-datum will be encoded so as to represent a data reliability classification, a data uncertainty classification, or the like. The meta-value need not be represented as a bit map. However, if the number of distinct data reliability classifications is small (e.g., less than or equal to the number of bits in a memory word), the bit map representation may be convenient and inexpensive to implement.

Depending on the circumstances, a data reliability parameter may be associated with a single datum, with a database tuple, or with a larger set of data.

Annotated Nullable Logic

Referring to Fig. 1, the present invention uses a form of computational logic herein called annotated nullable logic (ANL) for combining meta-data values.

In the context of the DBMS 100 shown in Fig. 1, the memory 114 of the system, which will typically include volatile memory (e.g., high speed random access memory) as well as non-volatile memory (e.g., disk storage), will typically contain:

- an operating system 130 for handling platform dependent tasks as well as typical operating system functions such as memory management, file system management, input/output tasks, and the like;
- application programs 132, which use the DBMS to store and retrieve data, perform transactions and resource management;
- a query parser 134, which parses and compiles queries received from application programs and users into execution plans;
- a query executer 136, which performs database access operations, including query predicate processing; and
- other programs not directly relevant to the present discussion.

The annotated nullable logic techniques of the present invention are primarily embodied in the query executer 136, and possibly also in the application programs 132, depending on the division of computational duties between the application programs 132 and the query engine 134/136 of the DBMS. Fig. 1 shows the query executer 136 as including an ANL module 138 for evaluating ANL expressions, also called meta-data expressions. Alternately, the ANL evaluation logic in the query executer 136 (and in the application program 132) may be implemented as in-line code within the query executer, instead of as a separate module or function.

As shown in Table 6, performing standard mathematical data combining operations (e.g., addition, subtraction, multiplication, division, concatenation of strings, and the like) on meta-data operands is accomplished by:

(A) performing the indicated mathematical data combining operation on the data-value portions of the meta-data operands, with the caveat that when one of the operands has an unknown or missing data value, the output data value is meaningless and may be represented as an unknown or missing data value; and

(B) performing a bitwise union of the meta-values of the operands.

When a mathematical data combining operation is performed on two non-annotated (ordinary, fully reliable) values, the result is another non-annotated value. More generally, when two values (also called operands) of the same reliability classification are combined, a result with the same reliability classification results. When a mathematical data combining operation is performed on a non-annotated value and an annotated value, the non-annotated value is explicitly or implicitly converted into an annotated value (with a reliable data meta-value) and then the two values are combined.

When two annotated operands of different degrees of reliability are mathematically combined, the result will be annotated with a bitwise union of the meta-values of two operands. It should be noted that when a value is annotated with two or more reliability classifications, in most application environments it is only the least reliable classification that is significant. For instance, if a value is annotated as being of both poor and medium reliability, the "medium" reliability classification is generally meaningless because the "poor" reliability classification is dominant. As a result, instead of combining annotations by forming a bitwise union of the annotations of the operands, it would be possible in alternate embodiments to combine annotations by generating a value representing the lowest reliability classification of the operands being combined. This methodology of selecting the lowest reliability classification, where appropriate, also has the advantage of working in embodiments where reliability classification annotations are encoded using a mechanism other than bitmaps.

The present invention does not require special handling for unknown and missing values. When combining two values, one of which is unknown (i.e., null in SQL parlance) the result is unknown. For instance, if both the "unknown" and "medium" reliability classification bits for

a result are set, the result is unknown because the "unknown" classification is dominant over the "medium" classification.

Table 6
Examples of Mathematical Operations using Annotated Nullable Logic

1) Example of adding two reliable numbers:
(5;00000)+(3;00000)=(8;00000)

2) Example of adding two numbers having same reliability classification:
(2.1;00001)+(3.3;00001)=(5.4;00001)

3) Examples of adding two numbers having different reliability classifications:
(7;00010)+(2;00100)=(9;00110)

(3;00100)+(N/A;10000)=(N/A;10100)

4) Example of concatenating two strings having different reliability classifications:
(NEWARK;00010)+(AIRPORT;00000)
=(NEWARK AIRPORT;00010)

In classical environments there are only two logical (or Boolean) values: True and False. In an ANL environment, there are still the same two logical values. Like values from all other domains (such as integers, real numbers, or strings), logical values in ANL may be annotated with meta-values. Logical values are often the result of comparisons between other values. For example, execution of an SQL query may require the expression x<4 to be evaluated for all the rows in a table (assuming the table has a column called x). For each row of the table, this expression will evaluate to either True or False. Alternatively, if the DBMS in question supports null values, the expression may sometimes (i.e., for some rows of the table) evaluate to a third logical value: unknown.

Table 7 contains some examples of comparisons of meta-data values (operands) using annotated nullable logic. The truth value part of the result generated by each comparison is equal to True, False or N/A (indicating that the truth value, if any, is meaningless). Generally, the meta-value for each comparison result is generated by performing a bitwise union (i.e., a bitwise logical OR'ing) of the meta-values of the operands. As in the example of combining meta-data operands, when a truth value generated by a meta-data comparison is annotated with

two or more reliability classifications, in most application environments it is only the least reliable classification that is significant.

Table 7
Examples of Numeric Comparison Operations using Annotated Nullable Logic

1) Example of comparing two reliable numbers:
(4;00000)<(5;00000)=(True;00000)

2) Examples of comparing two numbers having different reliability classifications:
(2;00001)=(2;00000)=(True;00001)

(3;00100)>(4;00010)=(False;00110)

4) Example of comparing reliable number with missing (not applicable) data value:
(6;00000)<(N/A;10000)=(N/A;10000)

Truth values can be combined by using the Boolean operators AND, OR and NOT (as well as other operators such as NAND, NOR, XOR and XNOR, that are derived from these basic operators). This is done routinely when formulating SQL database queries. As shown in Table 8, this is done in much the same way when using the annotated nullable logic technique of the present invention. The first example in Table 8 shows that "True AND True" yields a value of "True." However, since the operands for the AND operator are annotated, the result is annotated as well. When combining logical operands using the AND, NAND, XOR or XNOR operators, the meta-value for the result is generated by performing a bitwise union (i.e., a bitwise logical OR'ing) of the meta-values of the operands. When combining logical operands using the OR or NOR operators, the meta-value for the result is generated by selecting one of the operands' meta-values (e.g., in the case of the OR operation, the meta-value for the operand having the strongest truth value is selected).

Table 8
Examples of Combining Truth Values using Annotated Nullable Logic

1) Examples of using AND to combine two truth values having different reliability classifications:
(True;00010) AND (True;00001)=(True;00011)

(True;00010) AND (False;00001)=(False;00001)

(False;00010) AND (False;00001)=(False;00011)

2) Examples of using OR to combine two truth values having different reliability classifications:
(True ; 0 0 1 0 0) OR (True ; 0 0 0 0 0) = (True ; 0 0 0 0 0)

(True ; 0 0 1 0 0) OR (True ; 0 0 0 0 1) = (True ; 0 0 1 0 1)

(True ; 0 0 1 0 0) OR (False ; 0 0 0 0 1) = (True ; 0 0 1 0 0)

3) Example of using AND to combine a reliable truth value with a missing truth value:
(True ; 0 0 0 0 0) AND (N/A ; 1 0 0 0 0) = (N/A ; 1 0 0 0 0)

4) Example of using OR to combine a reliable truth value with missing truth value:
(False ; 0 0 0 0 0) OR (N/A ; 0 1 0 0 0) = (N/A ; 0 1 0 0 0)

When an unknown value is logically combined with a True or False value, an unknown truth value may or may not result, depending on the operation being performed. More particularly, Table 9 sets forth the rules that govern the combining of truth values in the preferred embodiment.

In summary, the annotated nullable logic combines first and second annotated values (k, A) and (m, B) using a predefined operator Op to generate an annotated result value (r, C):

$$(k, A)\ Op\ (m, B) = (r, C)$$

where k, m and r represent data values or truth values, and A, B and C represent data reliability values associated with the annotated first, second and result values, respectively. For some types of operations, such as numeric combining operations (addition, subtraction, multiplication, division, numeric value comparison), character manipulation operations and certain logic combining operations (e.g., AND, NAND, XOR, XNOR), the annotation bitmap for data reliability value C is generated by forming a bitwise union of the bitmaps for operands A and B. For other types of operations, such as OR'ing logical operands, the bitmap for data reliability value C is generated by selecting one operand's annotation bitmap (e.g., in the case of the OR operation, the annotation bitmap for operand having the strongest truth value is selected).

Table 9

Rules for Combining Truth Values using Annotated Nullable Logic

Notation:

Truth Values:

t means True
f means False
qt means Quasi True
qf means Quasi False
u means unknown

Strength of truth values: $f < qf < u < qt < t$.

l, m each mean a truth value that may be True, False, Quasi True, Quasi False, or unknown, with the following relative strength of truth: $l \leq m \leq$ True.

Annotation Meta-Values:

Ø means the empty set
A, B each mean a non-empty set of meta-values
N is a non-empty set of meta-values that indicate missing information (as opposed to unreliable information)
M is a non-empty set of meta-values that indicate unreliable information (as opposed to missing information)

Annotated Truth Values:

(t, Ø) means True
(f, Ø) means False
(f, N) and (t, N) both mean maybe (or unknown)
(t, M) means Quasi True, also denoted (qt, M)
(f, M) means Quasi False, also denoted (qf, M)

Basic ANL NOT operations:

NOT (t, Ø)	= (f, Ø)	(the inverse of True is False)
NOT (t, N)	= (f, N)	(the inverse of maybe is maybe)
NOT (f, Ø)	= (t, Ø)	(the inverse of False is True)
NOT (qt, M)	= (qf, M)	(the inverse of Quasi True is Quasi False)
NOT (qf, M)	= (qt, M)	(the inverse of Quasi False is Quasi True)

ANL Universal Bounds:

x OR False	= x or (f, Ø)	= x
x OR True	= x or (t, Ø)	= (t, Ø)
x AND False	= x AND (f, Ø)	= (f, Ø)
x AND True	= x AND (t, Ø)	= x

ANL OR and AND operations:

(m, A) OR (m, B)	= (m, A∪B)
(l, A) OR (m, B)	= (m, B)
(m, A) AND (m, B)	= (m, A∪B)
(l, A) AND (m, B)	= (l, A)

Commutativity:
(x and y are logical values, annotated or not)
x AND y = y AND x
x OR y = y OR x

Alternate Embodiments

In an alternate embodiment, for each combining and comparing operation where the preferred embodiment forms the union of the operands' meta-values, the ANL could instead generate a result meta-value representing the lowest reliability category associated with the operands. This would require the reliability categories to be ordered with regard to reliability to enable a determination of the lowest applicable reliability category.

The present invention could be used even in systems having just two levels of data reliability: reliable and unreliable, or reliable and missing. In such systems, meta data values would be annotated with a data reliability value that could be as short as a single bit (e.g., equal to 0 for reliable data and 1 for unreliable or missing data). However, the advantages of the present invention over systems using three valued logic are most readily appreciated when at least three distinct levels or categories of data reliability are used.

The present invention may be used in conjunction with a database management system that utilizes nested databases. For example, the nested database mechanisms disclosed in the Doctoral thesis by Ole Jørgen Anfindsen, "Apotram - an Application-Oriented Transaction Model," Chapter 5, "Nested Databases," UNIK Center for Technology at Kjeller, Department of Informatics, University of Oslo, Norway, Research Report 215, 1997, could be used in conjunction with the present invention, and the aforementioned Doctoral thesis is hereby incorporated by reference as background information.

The present invention may be implemented in DBMS's, transaction processing monitors, persistent programming languages, and concurrency control services for object resource brokerage systems. It may be distributed either in the form of such systems, or as a computer program product that is distributed on computer readable media such as magnetic data storage

media (i.e., disks or tapes), CD-ROM, or the like. A computer program product embodiment of the present invention may be distributed electronically, over the Internet or other communications network, as a computer data signal embodied in a carrier wave.

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A resource lock management system, comprising:

a lock data structure storing lock data representing granted and pending resource lock requests, wherein the data representing each granted and pending resource lock request includes: data indicating a resource to which access has been granted or requested, and an access mode associated with the resource lock request;

wherein a subset of the granted and pending resource lock requests are parameterized resource lock requests and the data representing each resource lock request in the subset further includes one or more parameter values indicating a data reliability classification associated with the resource lock request; and

a lock manager for evaluating, granting and denying resource lock requests, including determining when a resource lock request is unconditionally conflicting with any granted resource lock request, determining when the resource lock request is conditionally conflicting with any granted resource lock request, and evaluating the resource lock request with respect to each conditionally conflicting granted resource lock request by performing a predefined comparison of the parameter values for the resource lock request with the parameter values for each conditionally conflicting granted resource lock request.

2. The resource lock management system of claim 1, wherein the resource lock request is conditionally conflicting with a granted resource lock request only when (A) both the resource lock request and the granted resource lock request are parameterized resource lock requests and (B) the resource lock request is neither unconditionally conflicting nor unconditionally compatible with the granted resource lock request.

3. The resource lock management system of claim 2, wherein when resource lock request is a parameterized read lock request R(A), where A represents the parameter values for the parameterized read lock request, and the granted resource lock request is a parameterized write lock request W(B), where B represents the parameter values for the parameterized write lock request, the lock manager determines whether the resource lock request is conflicting with the granted resource lock request by determining whether or not B is a subset of A.

4. A resource lock management method, comprising the steps of:

storing lock data representing granted and pending resource lock requests, wherein the data representing each granted and pending resource lock request includes: data indicating a resource to which access has been granted or requested, and an access mode associated with the resource lock request;

wherein a subset of the granted and pending resource lock requests are parameterized resource lock requests and the data representing each resource lock request in the subset further includes one or more parameter values indicating a data reliability classification associated with the resource lock request; and

evaluating, granting and denying resource lock requests, including determining when a resource lock request is unconditionally conflicting with any granted resource lock request, determining when the resource lock request is conditionally conflicting with any granted resource lock request, and evaluating the resource lock request with respect to each conditionally conflicting granted resource by performing a predefined comparison of the parameter values for the resource lock request with the parameter values for each conditionally conflicting granted resource lock request.

5. The resource lock management method of claim 4, wherein the resource lock request is conditionally conflicting with a granted resource lock request only when (A) both the resource lock request and the granted resource lock request are parameterized resource lock requests and (B) the resource lock request is neither unconditionally conflicting nor unconditionally compatible with the granted resource lock request.

6. The resource lock management method of claim 5, wherein when resource lock request is a parameterized read lock request R(A), where A represents the parameter values for the parameterized read lock request, and the granted resource lock request is a parameterized write lock request W(B), where B represents the parameter values for the parameterized write lock request, the resource lock request is evaluated with respect to the granted resource lock request by determining whether or not B is a subset of A.

7. A computer program product for use in conjunction with a computer system, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

instructions for storing lock data representing granted and pending resource lock requests, wherein the data representing each granted and pending resource lock request includes: data indicating a resource to which access has been granted or requested, and an access mode associated with the resource lock request;

wherein a subset of the granted and pending resource lock requests are parameterized resource lock requests and the data representing each resource lock request in the subset further includes one or more parameter values indicating a classification associated with the resource lock request; and

lock manager means for evaluating, granting and denying resource lock requests, including determining when a resource lock request is unconditionally conflicting with any granted resource lock request, determining when the resource lock request is conditionally conflicting with any granted resource lock request, and evaluating the resource lock request with respect to each conditionally conflicting granted resource lock request by performing a predefined comparison of the parameter values for the resource lock request with the parameter values for each conditionally conflicting granted resource lock request.

8. The computer program product of claim 7, wherein the resource lock request is conditionally conflicting with a granted resource lock request only when (A) both the resource lock request and the granted resource lock request are parameterized resource lock requests and (B) the resource lock request is neither unconditionally conflicting nor unconditionally compatible with the granted resource lock request.

9. The resource lock management system of claim 8, wherein when resource lock request is a parameterized read lock request R(A), where A represents the parameter values for the parameterized read lock request, and the granted resource lock request is a parameterized write lock request W(B), where B represents the parameter values for the parameterized write lock request, the lock manager determines whether the resource lock request is conflicting with the granted resource lock request by determining whether or not B is a subset of A.

10. A data management system, comprising:

a database, the database storing information, a subset of the stored information comprising annotated values, each annotated value having a stored data value and an associated data reliability value, wherein the data reliability value is a member of a domain that

includes at least three distinct data reliability values corresponding to at least three distinct data reliability classifications that can apply to the annotated values stored in the database; and

a query executer for executing queries requesting access to information stored in the database;

the query executer including annotated nullable logic for evaluating expressions containing at least one annotated value as an operand, wherein the annotated nullable logic includes logic for combining annotated values, for comparing annotated values so as to generate annotated truth values, and for combining annotated truth values in accordance with a predefined set of rules and so as to preserve relevant ones of the data reliability values associated with the annotated values and annotated truth values that have been combined and compared.

11. The data management system of claim 10, wherein

the data reliability values associated with the annotated values are each encoded as a reliability category bitmap having a plurality of bits that each correspond to a respective one of the data reliability classifications;

the annotated nullable logic includes means for combining first and second annotated values (k, A) and (m, B) using a predefined operator Op to generate an annotated result value (r, C):

$$(k, A)\ Op\ (m, B) = (r, C)$$

where k, m and r represent data values or truth values, and A, B and C represent data reliability values associated with the annotated first, second and result values, respectively; and C is generated by the annotated nullable logic by generating a union of A and B.

12. The data management system of claim 11, wherein

one of the data reliability classifications represents fully reliable data and at least two of the distinct data reliability classifications represent two distinct classifications of missing data or less than fully reliable data.

13. A data processing product, comprising:

means for storing information, a subset of the stored information comprising annotated values, each annotated value having a stored data value and an associated data reliability value, wherein the data reliability value is a member of a domain that includes at least three distinct

data reliability values corresponding to at least three distinct data reliability classifications that can apply to the annotated values stored in the database; and

means for performing information processing using the stored information;

the information processing means including annotated nullable logic for evaluating expressions containing at least one annotated value as an operand, wherein the annotated nullable logic includes logic for combining annotated values, for comparing annotated values so as to generate annotated truth values, and for combining annotated truth values in accordance with a predefined set of rules and so as to preserve relevant ones of the data reliability values associated with the annotated values and annotated truth values that have been combined and compared.

14. The data processing product of claim 13, wherein

the data reliability values associated with the annotated values are each encoded as a reliability category bitmap having a plurality of bits that each correspond to a respective one of the data reliability classifications;

the annotated nullable logic includes means for combining first and second annotated values (k, A) and (m, B) using a predefined operator Op to generate an annotated result value (r, C):

$$(k, A)\ Op\ (m, B) = (r, C)$$

where k, m and r represent data values or truth values, and A, B and C represent data reliability values associated with the annotated first, second and result values, respectively; and C is generated by the annotated nullable logic by generating a union of A and B.

15. The data processing product of claim 14, wherein

one of the data reliability classifications represents fully reliable data and at least two of the distinct data reliability classifications represent two distinct classifications of missing data or less than fully reliable data.

16. A data processing method, comprising the steps of:

storing information, a subset of the stored information comprising annotated values, each annotated value having a stored data value and an associated data reliability value, wherein the data reliability value is a member of a domain that includes at least three distinct data reliability values corresponding to at least three distinct data reliability classifications that can apply to the annotated values stored in the database; and

performing information processing using the stored information;

the information processing step including combining annotated values to generate new annotated value, comparing annotated values so as to generate annotated truth values, and combining annotated truth values in accordance with a predefined set of rules and so as to preserve relevant ones of the data reliability values associated with the annotated values and annotated truth values that have been combined and compared.

17. The data processing method of claim 16, wherein

the data reliability values associated with the annotated values are each encoded as a reliability category bitmap having a plurality of bits that each correspond to a respective one of the data reliability classifications;

the annotated nullable logic includes means for combining first and second annotated values (k, A) and (m, B) using a predefined operator Op to generate an annotated result value (r, C):

$$(k, A)\ Op\ (m, B) = (r, C)$$

where k, m and r represent data values or truth values, and A, B and C represent data reliability values associated with the annotated first, second and result values, respectively; and C is generated by the annotated nullable logic by generating a union of A and B.

18. The data processing method of claim 17, wherein one of the data reliability classifications represents fully reliable data and at least two of the distinct data reliability classifications represent two distinct classifications of missing data or less than fully reliable data.

19. A computer program product for use in conjunction with a computer system, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

instructions for storing information in a storage device associated with the computer system, a subset of the stored information comprising annotated values, each annotated value having a stored data value and an associated data reliability value, wherein the data reliability value is a member of a domain that includes at least three distinct data reliability values corresponding to at least three distinct data reliability classifications that can apply to the annotated values stored in the database; and

instructions for performing information processing using the stored information;

the information processing instructions including annotated nullable logic instructions for evaluating expressions containing at least one annotated value as an operand, wherein the annotated nullable logic includes logic for combining annotated values, for comparing annotated values so as to generate annotated truth values, and for combining annotated truth values in accordance with a predefined set of rules and so as to preserve relevant ones of the data reliability values associated with the annotated values and annotated truth values that have been combined and compared.

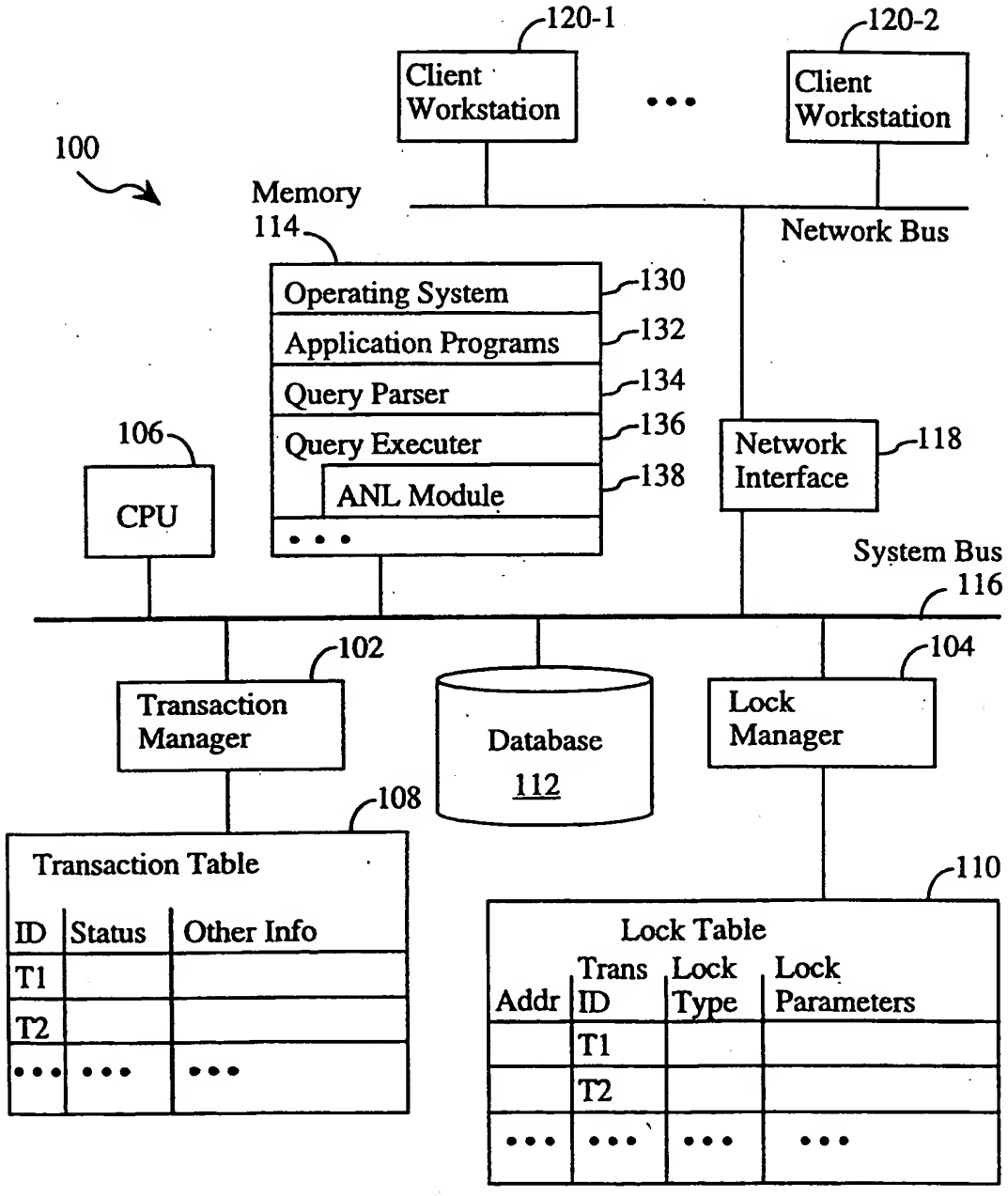

120-1
Client Workstation
120-2
Client Workstation
100
Memory
114
Network Bus
Operating System
130
Application Programs
132
Query Parser
134
Query Executer
136
ANL Module
138
106
CPU
Network Interface
118
System Bus
116
102
Transaction Manager
Database
112
104
Lock Manager
108
Transaction Table
ID
Status
Other Info
T1
T2
110
Lock Table
Addr
Trans ID
Lock Type
Lock Parameters
T1
T2

FIG. 1

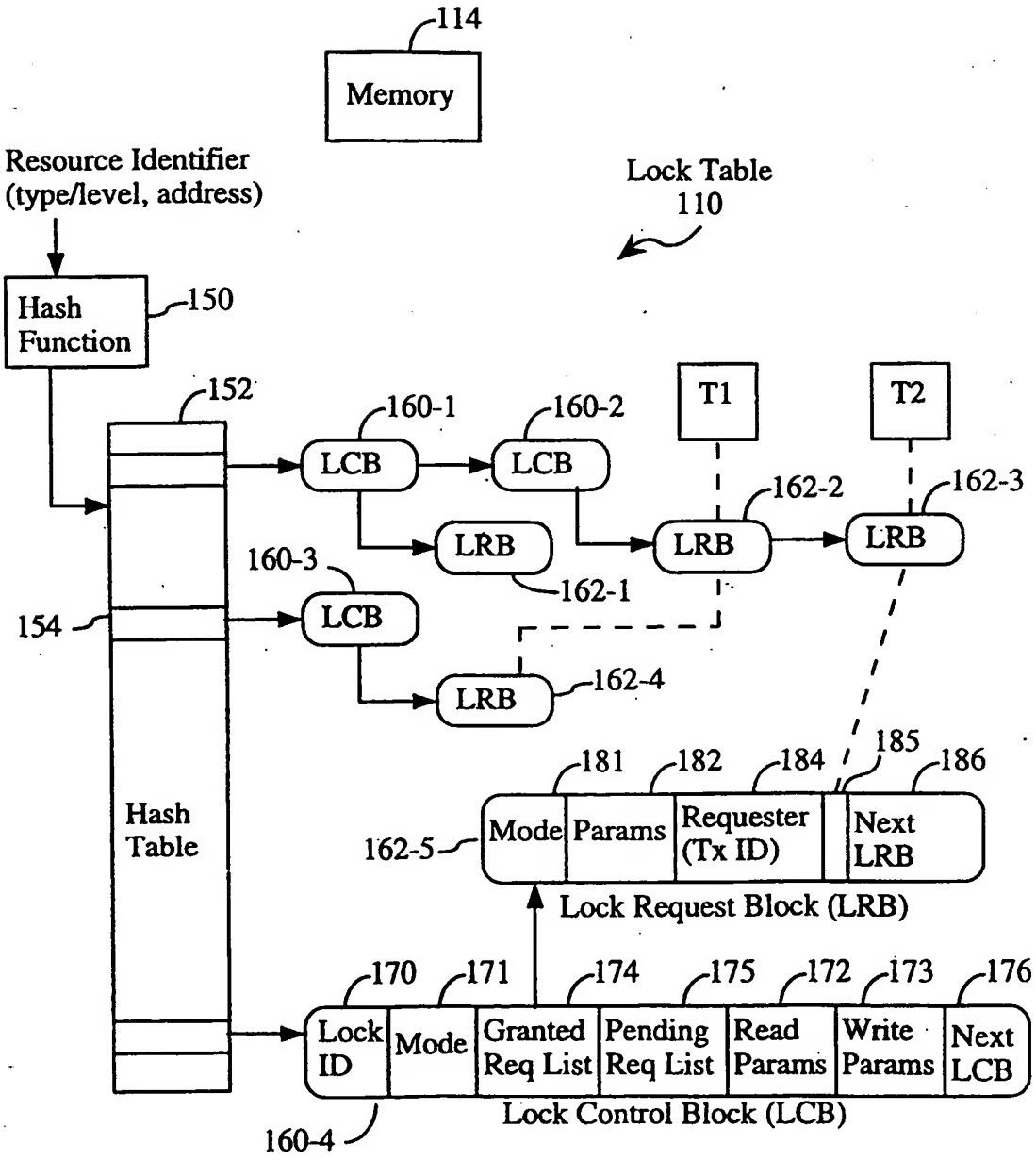

114
Memory
Resource Identifier
(type/level, address)
Lock Table
110
Hash
Function
150
152
160-1
160-2
T1
T2
LCB
LCB
LRB
LRB
LRB
162-2
162-3
160-3
LCB
162-1
154
LRB
162-4
185
181
182
184
186
Hash
Table
162-5
Mode
Params
Requester
(Tx ID)
Next
LRB
Lock Request Block (LRB)
170
171
174
175
172
173
176
Lock
ID
Mode
Granted
Req List
Pending
Req List
Read
Params
Write
Params
Next
LCB
160-4
Lock Control Block (LCB)

FIG. 2

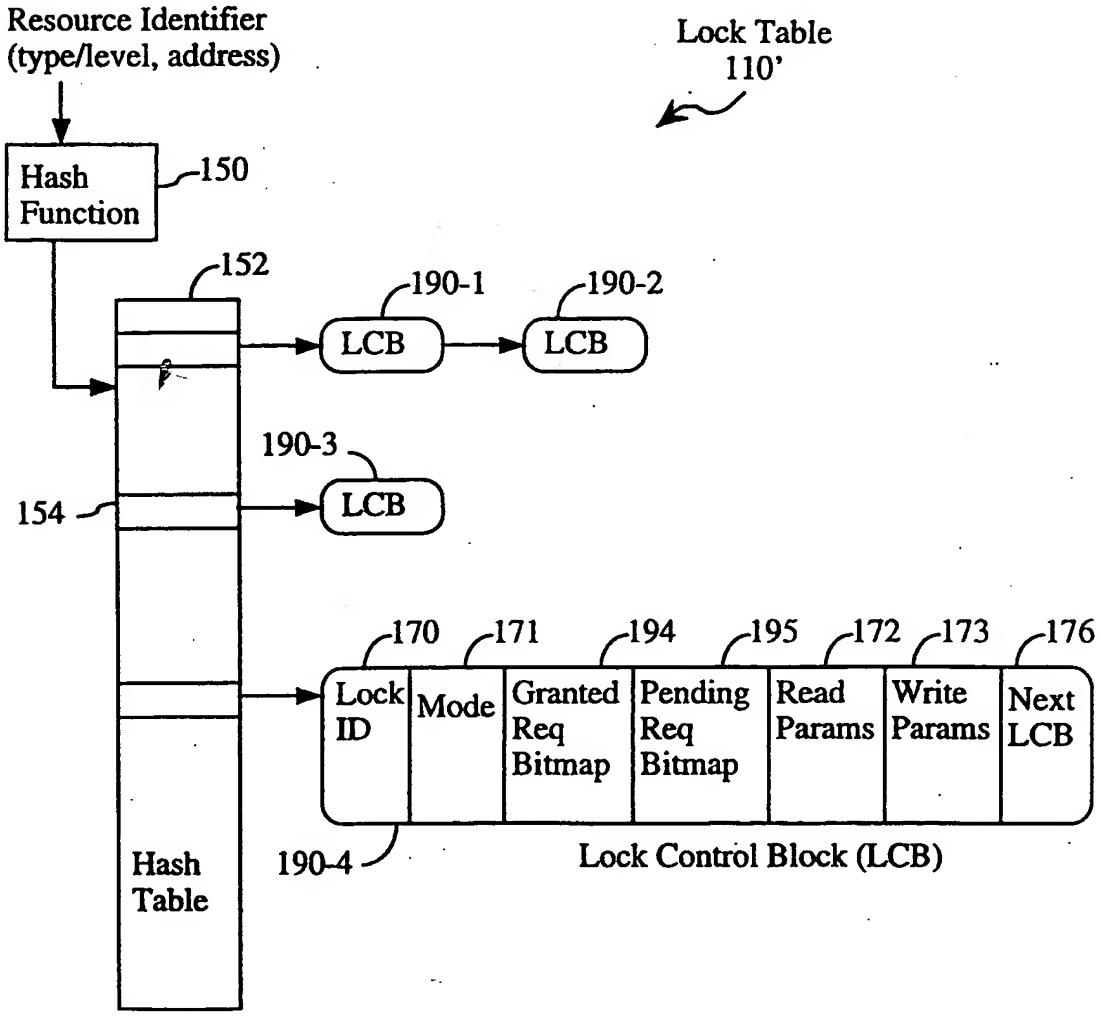

Resource Identifier
(type/level, address)
Lock Table
110'
Hash
Function
150
152
190-1
190-2
LCB
LCB
190-3
LCB
154
170
171
194
195
172
173
176
Lock
ID
Mode
Granted
Req
Bitmap
Pending
Req
Bitmap
Read
Params
Write
Params
Next
LCB
Hash
Table
190-4
Lock Control Block (LCB)

FIG. 3

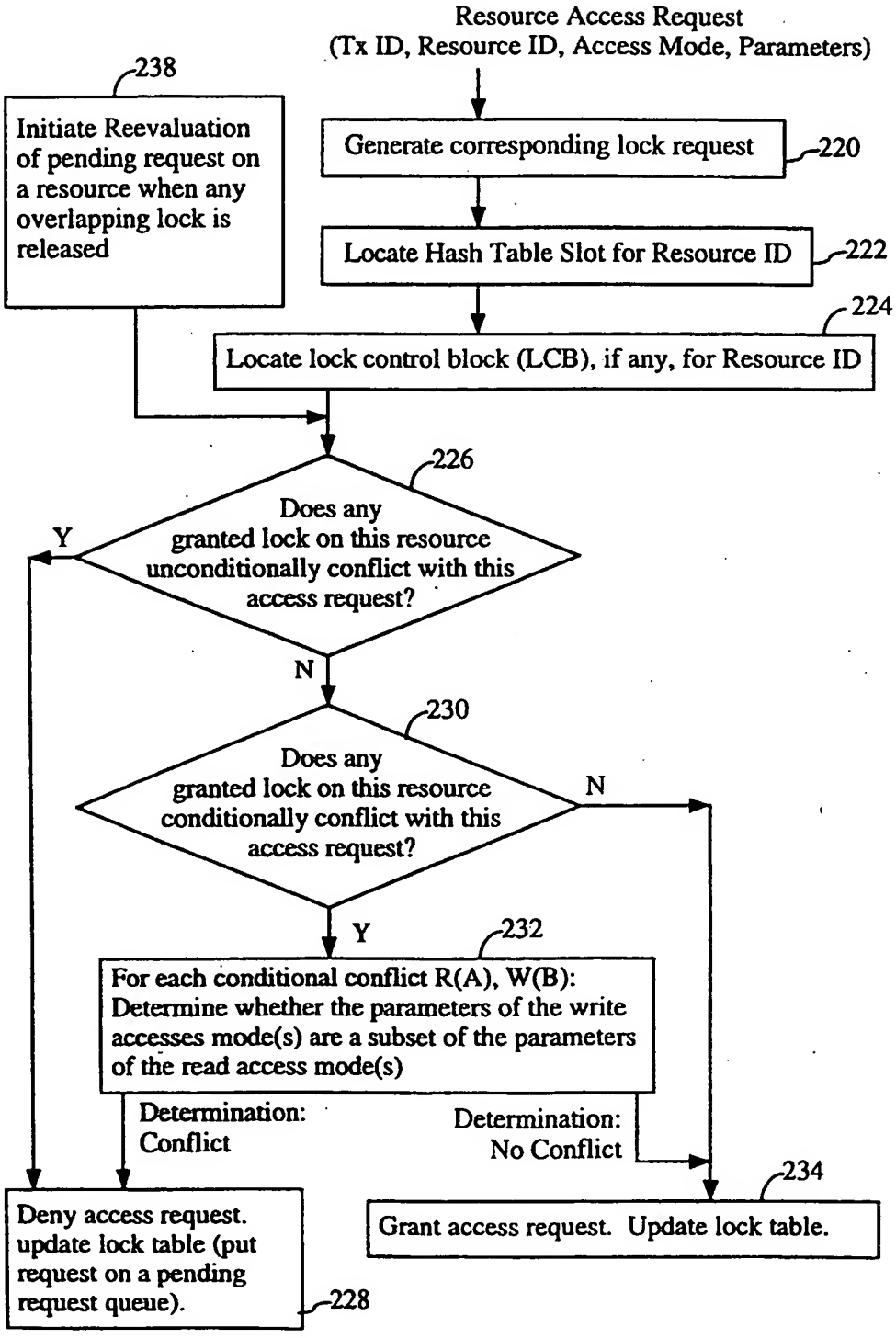

Resource Access Request
(Tx ID, Resource ID, Access Mode, Parameters)
238
Initiate Reevaluation of pending request on a resource when any overlapping lock is released
Generate corresponding lock request
220
Locate Hash Table Slot for Resource ID
222
224
Locate lock control block (LCB), if any, for Resource ID
226
Does any granted lock on this resource unconditionally conflict with this access request?
Y
N
230
Does any granted lock on this resource conditionally conflict with this access request?
N
Y
232
For each conditional conflict R(A), W(B): Determine whether the parameters of the write accesses mode(s) are a subset of the parameters of the read access mode(s)
Determination: Conflict
Determination: No Conflict
234
Grant access request. Update lock table.
Deny access request. update lock table (put request on a pending request queue).
228

FIG. 4

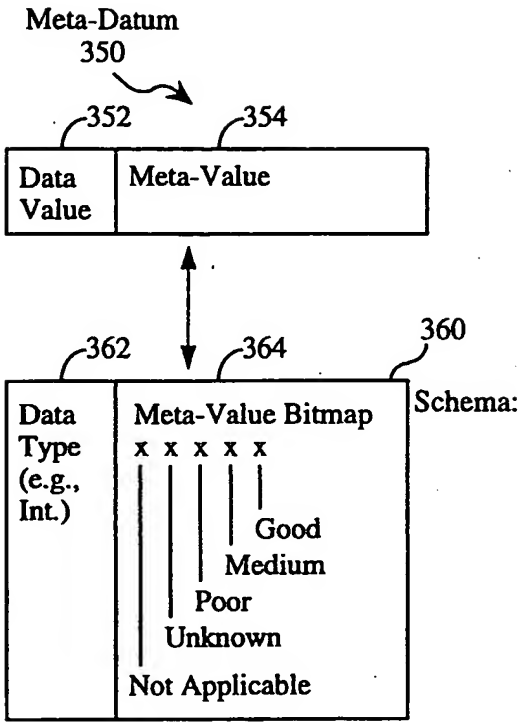


FIG. 5A

5	0 0 0 0 0	Integer 5, 100% Reliable
3.14	0 0 0 0 1	Real value 3.14, Good Reliability
London	0 0 0 1 0	String "London", Medium Reliability
N/A	0 1 0 0 0	An Unknown Value

FIG. 5B

INTERNATIONAL SEARCH REPORT

Inter onal Application No
PCT/NO 99/00018

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	HULL R: "Global predicate-based locks in long-running transactions" PROCEEDINGS. FIRST INTERNATIONAL ENTERPRISE DISTRIBUTED OBJECT COMPUTING WORKSHOP (CAT. NO.97TB100130), PROCEEDINGS FIRST INTERNATIONAL ENTERPRISE DISTRIBUTED OBJECT COMPUTING WORKSHOP, GOLD COAST, QLD., AUSTRALIA, 24-26 OCT. 1997, pages 197-206, XP002106675 ISBN 0-8186-8031-8, 1997, Los Alamitos, CA, USA, IEEE Comput. Soc, USA see the whole document ---	1-19
A	EP 0 817 019 A (IBM) 7 January 1998 see the whole document --- -/--	1-19

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 June 1999	02/07/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Katerbau, R

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inter ›nal Application No
PCT/NO 99/00018

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 432 926 A (CITRON ANDREW P ET AL) 11 July 1995 see the whole document	1-19
A	US 5 692 178 A (SHAUGHNESSY STEVEN T) 25 November 1997 see the whole document	1-19

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

..formation on patent family members

Inter ›nal Application No
PCT/NO 99/00018

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0817019 A	07-01-1998	JP 10069418 A	10-03-1998
US 5432926 A	11-07-1995	JP 2708357 B	04-02-1998
		JP 6215032 A	05-08-1994
US 5692178 A	25-11-1997	US 5555388 A	10-09-1996
		CA 2101569 A	21-02-1994
		EP 0588502 A	23-03-1994
		US 5561793 A	01-10-1996